Discrete Applied Mathematics 42 (1993) 271–278
North-Holland

Computing the throughput of a network with dedicated lines

Christos H. Papadimitriou*

Department of Computer Science, University of California at San Diego, La Jolla, CA, USA

Paolo Serafini

Department of Computer Science, University of Udine, Udine, Italy

Mihalis Yannakakis

AT&T Bell Laboratories, Murray Hill, NJ, USA

Received 15 June 1990
Revised 1 December 1990

Abstract

Papadimitriou, C.H., P. Serafini and M. Yannakakis, Computing the throughput of a network with dedicated lines, Discrete Applied Mathematics 42 (1993) 271–278.

Suppose that we wish to transmit many messages from a node of a network to another, and the delays of the edges of the network are known. Once a message has been sent along an edge, the edge cannot be used to transmit another message until the first message arrives at the other end. We wish to estimate the throughput of such a network, that is, the number of messages from the source to the destination that can be transmitted within a given time T. We show that this throughput is hard to compute for finite values of T, but can be estimated asymptotically using flow techniques.

1. Introduction

Consider a directed graph $G=(V,E)$ with positive integer weights on its edges, and two specified nodes in it, the *source* s and the *destination* d. The graph is a

Correspondence to: Professor C.H. Papadimitriou, Department of Computer Science, University of California at San Diego, La Jolla, CA 92093, USA.

* Research supported by the National Science Foundation and the ESPRIT Project ALCOM.

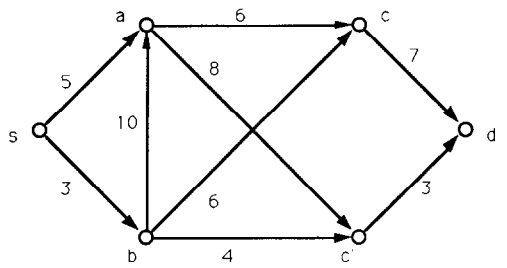

Fig. 1.

communication network, and the weight of an arc (x, y), $w(x, y)$, stands for the time it takes for a message to reach y after it is forwarded by x (see Fig. 1). Optimization problems on such networks, such as the shortest path problem, have been studied extensively in the past (see, for example, [3, 6]).

When we wish to send a single message from s to d, we should solve the shortest path problem and send the message along the resulting path. Suppose however that we wish to send *many messages* from s to d along the network. *Furthermore, once a message is sent, assume that no other message can be sent along the same line, until the first message has arrived at the other end.* This restriction would arise, for example, when the protocol requires that an acknowledgement must be received by the sender before the communication can proceed further (in this case, the weight of an arc would be twice its delay). We wish to maximize the number of messages that can be sent from s to d within a fixed time interval T under this regime. Notice that we are assuming that a message causes no delay at the sending or receiving node; naturally, such delays can be captured by replacing each node by an appropriate bipartite graph.

For an application from a domain other than communication, imagine a manufacturing system containing machines of varying speeds and capabilities. The machines can be represented by the edges of the network, with weight equal to the processing time. Any path from s to d is an acceptable way of manufacturing a unit of the product.

This natural variant of the shortest path problem has apparently not been studied before. One realizes quickly that it is a rather intricate problem. For example, in Fig. 1 we should obviously send messages as often as we can along arcs (s, a) and (s, b). However, when these messages arrive, which one of the arcs out of a and b should be preferred, if available? Should we ever wait for an arc to become free, if another one out of the same node is currently free?

We show in Section 2 that computing the maximum possible number $\mu(T)$ of messages that have arrived at d by time T is a strongly NP-hard problem; it is in PSPACE, and we conjecture that it is PSPACE-complete. However, we also show that we can compute the optimum asymptotic ratio $\mu = \lim_{T \to \infty} \mu(T)/T$ in polynomial time, using maximum flow techniques. Our result exploits the following

intuition: If the delay of a line is w seconds, this is tantamount to the line having a *capacity* of $1/w$ messages per second. A maximum flow problem results, which is used to maximize the throughput. But it takes some nontrivial arguments to make this intuition work. The resulting schedule is extremely regular and natural.

There is another interesting version of this problem, suggested to us by T.C. Hu (in fact, his question started us on this area). Imagine that, on sending a message out of s to d, we must decide beforehand the full path it will follow, and *we cannot use for another message any arc on this path until the message has arrived to d.* This would correspond to the "circuit switching" (as opposed to "packet switching") style of network communication, again when we insist on dedicated lines (or acknowledgements). As before, we wish to maximize the asymptotic throughput. Graph theoretically, we wish to maximize over all sets P of disjoint paths from s to d the quantity $\mu(P)=\sum_{p_j\in P} 1/w(p_j)$, where $w(p_j)$ is the total weight of path p_j. We show in Section 3 that this problem is NP-complete.

2. Exact vs. asymptotic throughput

Given a weighted network (G,s,d,w) and an integer T, a *schedule* is, intuitively, a plausible plan for sending messages in the network. Formally, a schedule σ is a set $\{(e_i,t_i)\colon i=1,\ldots,n\}$ of arc-time pairs, $\sigma\subset E\times\{0,\ldots,T\}$. Intuitively, each element (e_i,t_i) in σ denotes the event that at time t_i (without loss of generality, an integer) a message is sent from x to y along edge $e_i=(x,y)$. The following must hold: (1) If there are two distinct events (e,t) and (e,t'), then $|t-t'|\geq w(e)$, that is, no edge should be used again before the last message sent on it has arrived at the destination; (2) at each time t and each node $x\neq s$, $|\{((x,y),t')\in\sigma\colon y\in V, t'\leq t\}|\geq|\{((y,x),t')\in\sigma\colon y\in V, t'+w(y,x)\leq t\}|$, that is, each node (except for s) at all times must have sent out no more messages than it has received.

A schedule σ is evaluated by its *throughput*, that is the number of messages it delivers to d by time T, that is, $\mu(\sigma)=|\{((x,d),t)\in\sigma\colon x\in V, t+w(x,d)\leq T\}|$. The throughput of the network is the function $\mu(T)=\min_\sigma \mu(\sigma)$, the minimum taken over all legal schedules.

We shall study the computational problem of computing the throughput and asymptotic throughput of a network. In particular, define THROUGHPUT to be the following problem: "Given a network $((V,E),s,d,w)$ and integers T and M (in binary), is $\mu(T)\geq M$?" Also, define ASYMPTOTIC THROUGHPUT to be the following problem: Given a network $((V,E),s,d,w)$, compute $\lim\sup_{T\to\infty}(\mu(T)/T)$. We shall find, perhaps surprisingly, that the latter problem is much easier to solve than the former.

We first show the following upper bound for the complexity of THROUGHPUT:

Theorem 1. *THROUGHPUT can be solved in polynomial space.*

Proof. We shall describe a polynomial space *nondeterministic* algorithm for deciding whether $\mu(T) \geq M$; a deterministic algorithm follows from Savitch's theorem [7]. Our algorithm uses nondeterminism to "guess" the optimal schedule. Since the optimal schedule is in general an exponentially long object, our algorithm guesses it one pair (e, t) at a time, in order of nondecreasing t. Our algorithm works in T stages. At stage t, it nondeterministically updates the *state* $S(t)$ of the network. $S(t)$ contains necessary information for guessing and simulating the schedule. In particular, for each node x we maintain $n(x, t)$, a number between 0 and T denoting the number of messages waiting at this node at time t. For each arc (x, y) we maintain an integer $-1 \leq p(x, y, t) \leq w(x, y)$, the *progress* made by the message currently traversing arc (x, y). If $p(x, y, t)$ becomes $w(x, y)$, it is reinitialized to -1 (no message in transit) and $n(y, t)$ is increased by one. Besides this updating, at each time t we may decide to add some pair $((x, y), t)$ to the schedule. This has the effect of decreasing $n(x, t)$ by one (assuming it was positive), and setting $p(x, y, t)$ to zero (assuming it was -1).

Initially, all $n(x, 0)$ are zero, except that $n(s, 0) = T|E|$ (enough messages for the whole process), and all $p(x, y, 0)$ are -1. We accept if at the end $n(d, T) \geq M$. Obviously, the process requires only polynomial space. □

We conjecture that THROUGHPUT is PSPACE-complete. All we can prove is the following:

Theorem 2. *THROUGHPUT is* NP-*hard.*

Proof. We shall reduce the NP-complete problem THREE PARTITION to it. In this problem we are given three sets of n integers each, $\{a_1, \ldots, a_n\}, \{b_1, \ldots, b_n\}$, and $\{c_1, \ldots, c_n\}$, each greater than $B/4$, where $B = (1/n) \sum_{i=1}^{n} (a_i + b_i + c_i)$. We are asked whether there are two permutations β and γ of $\{1, \ldots, n\}$ such that all sums $a_i + b_{\beta(i)} + c_{\gamma(i)}, i = 1, \ldots, n$, are equal to B. This is not the standard version of the problem mentioned in [1], but it is easy to see that it is also strongly NP-complete.

Given an instance of THREE PARTITION, we construct the following network G. G has only four nodes s, b, c, d. Between s and b there are n parallel arcs of weights a_i; between b and c there are n parallel arcs of weights b_i; and similarly for c, d and c_i (see Fig. 2). If we insist on forbidding parallel edges, this is easy to arrange by establishing midpoints at all these arcs. We take T to be equal to B and $M = n$. It is easy to check that there is a schedule that delivers n messages to d by time B if and only if the original instance of THREE PARTITION had a solution. The correspondence between permutations and schedules is this: $\beta(i)$ is the index of the arc in the second layer that will take the message originally sent through arc i of the first layer, and similarly for $\gamma(i)$; the correspondence goes both ways. □

The rest of this section is devoted to showing that, in contrast, ASYMPTOTIC THROUGHPUT can be solved in polynomial time. From $((V, E), s, d, w)$ we construct

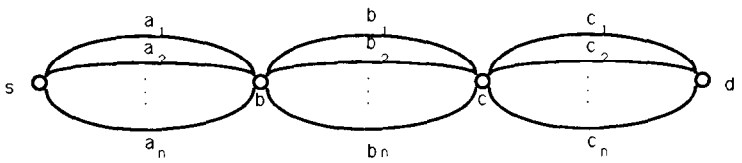

Fig. 2. Transformation from THREE PARTITION.

a flow network by replacing weights $w(x, y)$ by the capacity $1/w(x, y)$. Let F be the value of the maximum flow in the resulting network. Let $\mu = \limsup_{T\to\infty} (\mu(T)/T)$.

Lemma 3. $\mu \leq F$.

Proof. By the max-flow min-cut theorem, there is a cut C in the flow network that separates d from s and has total capacity F. Consider an edge (x, y) in this cut. Any schedule σ for time t cannot push through (x, y) more than $T/w(x, y)$ messages, and thus σ delivers to d no more than $\sum_{(x,y)\in C} T/w(x, y) = T \cdot F$ messages. Therefore, $\mu(T)/T \leq F$, and thus $\mu = \limsup_{T\to\infty} (\mu(T)/T) \leq F$. □

The following result states that the bound F can be actually achieved asymptotically. This is rather surprising, since the obvious algorithm for achieving F (send a message along arc (x, y) every $1/f(x, y)$ seconds, where $f(x, y)$ is the value of the optimal flow) runs into trouble because of unavailable messages at x. Our proof uses an asymptotically insignificant initial phase that creates a "cache" of messages at each node; the calculation shows that these caches need only be bounded.

Lemma 4. $\mu \geq F$.

Proof. Our scheduling algorithm is simple: We run an initialization phase whose goal is to create a stock of $n_0(x)$ messages at each node x. We then run the obvious algorithm: Let $f(x, y)$ be the value of the optimal flow through arc (x, y). We send a message along arc (x, y) every $1/f(x, y)$ time units. It is clear that, if the initialization phase is finite, the throughput $\mu(T)$ of this algorithm will asymptotically tend to F.

It remains to prove that a bounded number $n_0(x)$ messages will be sufficient for any length T of operations. Consider a node x, with incoming arcs (y_i, x), $i = 1, \ldots, p$, and outgoing arcs $(x, z_i), i = 1, \ldots, q$. At time T, the number of messages that have been sent out from node x is $\sum_{i=1}^{q} \lceil T/(1/f(x, z_i)) \rceil$, whereas the number of messages that have been received is $\sum_{i=1}^{p} \lfloor T/(1/f(y_i, x)) \rfloor$. Thus, we only need to have an initial stock of $n_0(x) = \sum_{i=1}^{q} \lceil Tf(x, z_i) \rceil - \sum_{i=1}^{p} \lfloor Tf(y_i, x) \rfloor$, which is at most $(\sum_{i=1}^{q} Tf(x, z_i) + q) - (\sum_{i=1}^{p} Tf(y_i, x) - p)$, or $p + q$, since f is a legitimate flow and thus $\sum_{i=1}^{q} f(x, z_i) = \sum_{i=1}^{p} f(y_i, x)$. It follows that the initial phase need accumulate at each node x a number $n_0(x)$ of messages equal to its indegree plus its outdegree, a number independent of T. The result follows. □

Theorem 5. *ASYMPTOTIC THROUGHPUT can be solved in polynomial time.*

Proof. From Lemmas 3 and 4 we conclude that the value of the asymptotic throughput μ can be computed as the value of the maximum flow of the network. This can be done in $O(n^3)$ time [4], or even faster for sparse networks. □

It is quite easy to see that both the positive and negative results in this section can be extended to the case in which messages can be transmitted in both directions of (all or some of) the edges of the graph.

It turns out that we can extend the results in another interesting direction: the case of *explicit acknowledgement*. Suppose that, once a message arrives at d, an acknowledgement to s is generated and sent. These new messages must travel from d back to s, possibly interfering with other messages in transit, reducing the throughput of the network. The throughput is now defined as the number of acknowledgements that have arrived at s by time T. The problem is easily seen to be NP-hard (just add n edges from d to s, with delay B, to the network in the NP-hardness construction). Also, the appropriate flow formulation is now a *2-commodity flow problem*, which can be solved in polynomial time, see [2]. Its solution is a lower bound to the asymptotic throughput of the network, while it again suggests a simple schedule with asymptotically optimal performance. For the details of the generalizations in these two paragraphs, the reader is referred to [5].

3. Dedicated paths

Suppose now that we must select a few node-disjoint paths from s to d, and always route messages along one of these paths. Once a message is sent along such a path, the next message along the path must wait for the first one to arrive at d. It is easy to see that the asymptotic throughput of this style of communication is $\mu(P) = \sum_{p_j \in P} 1/w(p_j)$, where $P = \{p_1, \ldots, p_k\}$ is the set of node-disjoint paths from s to d used, and $w(p_j)$ stands for the total weight of the path.

Thus we are led to the following unusual combinatorial optimization problem: INVERSE PATH SUM: "Given a weighted network and integer B, is there a set $P = \{p_1, \ldots, p_k\}$ of node-disjoint paths from s to d such that $\sum_{j=1}^{k} 1/w(p_j) \geq B$?"

Theorem 6. *INVERSE PATH SUM is* NP*-complete.*

Proof. We shall reduce the NP-complete problem 3-SAT to it. We are given a set $\{x_1, \ldots, x_n\}$ of variables and a set $\{C_1, \ldots, C_m\}$ of clauses, with each clause consisting of three literals. We are asked whether there is a truth assignment that satisfies all clauses. We construct a network $((V, E), s, d, w)$ and a constant B such that there is a set of node-disjoint paths P with $\sum_{j=1}^{k} 1/w(p_j) \geq B$ if and only if a satisfying truth assignment exists.

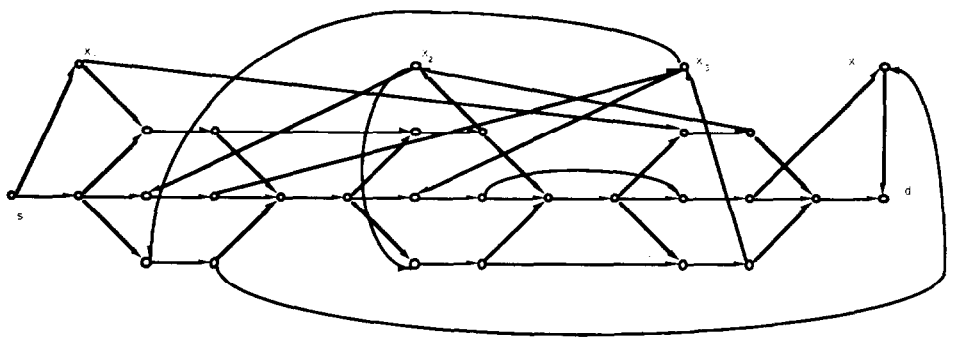

Fig. 3. Transformation from $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_3 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_2)$.

The network is shown in Fig. 3. It consists of a part related to the clauses with edge weights one, and one related to the variables. For each clause we have three parallel paths of length three, all connected in series from s to d. Each occurrence of a literal in a clause has in this way a particular edge (the middle edge of one path) associated with it. For each variable we have another subgraph, and all these subgraphs are again connected in series from s to d. All these edges have weight M. Thus, there are many paths of length $4m+1$ from s to d, but of course no two of them are disjoint (since they all share the edges from one clause to the next).

The subgraph associated with variable x_i consists of two parallel paths, one going through all occurrences of x_i in the clauses, and the other through all occurrences of $\bar{x}_i$. All edges in the variable part have weight M, a huge number (say, $M > 100mn$). This concludes the construction of the network. We take

$$B = \frac{1}{4m+1} + \frac{1}{M^2}.$$

Suppose that there is a truth assignment that satisfies all clauses; we shall exhibit two paths whose sum of inverse lengths exceed B. The first path is of length $4m+1$, and traverses the clause part from s to d *taking at each clause the path corresponding to a true literal* under the satisfying truth assignment (we know there is at least one in each clause). The second path traverses the variable part, and at each variable it follows *the path corresponding to the opposite of the truth value* of this variable by the satisfying truth assignment; this assures us that the path will be kept edge disjoint from the first one. Its total length is less than M^2, and thus the bound is attained.

Conversely, suppose that a set P of paths with total sum of inverse lengths at least B exists. At least one of the paths in P must contain no edge of weight M, since B is larger than any possible multiple of $1/M$. Thus, one of these paths, call it p_1, must belong totally in the clause part of the network. This path alone is not enough to achieve B, and so another path must exist which is node disjoint from p_1, call it p_2. Since all nodes on the clause side except for the intermediate ones have been traversed by p_1, p_2 is confined to the variable part, and thus it traverses one of the

two parallel paths for each variable. Think of the fact that p_2 traverses the path corresponding to a literal to mean that this literal is false. This obviously defines a truth assignment. Furthermore, this truth assignment must satisfy all clauses, since p_2 has traversed all clauses, and is edge disjoint with p_2. □

References

[1] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).

[2] A. Itai, Two-commodity flow, J. ACM 25 (1978) 596–611.

[3] E.L. Lawler, Combinatorial Optimization: Networks and Matroids (Holt, Rinehart & Winston, New York, 1977).

[4] V.M. Malhotra, M.P. Kumar and S.N. Maheshwari, An $O(|V|^3)$ algorithm for finding maximum flows in networks, Inform. Process. Lett. 7 (1978) 277–278.

[5] C.H. Papadimitriou, P. Serafini and M. Yannakakis, Computing the throughput of a network with dedicated lines, Tech. Rept., University of Udine, Udine.

[6] C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[7] W.J. Savitch, Relationships between nondeterministic and deterministic space complexities, J. Comput. System Sci. 4 (1970) 177–192.